EE2703 : Applied Programming Lab
Assignment 9
DFT of Non-Periodic Signals

Sai Shashank GP
EE20B040

May 14, 2022

0.1 Introduction

In this assignment, we are going to use the techniques of DFT we have seen in Assignment 8 for Aperiodic signals and try some concepts like **windowing** etc... to make the DFTs better. We will be using **pylab** for plotting the 3D and 2D figures, **numpy** for array and vector operations

0.2 Assignment

0.2.1 Prerequisites

We are going to import required libraries and define required classes and helper functions for the assignment

```
# Import useful libraries
from pylab import *
import numpy as np

# Defining some important signals' classes so that they come in handy
class sinsq2t:
    def __init__(self, t):
        self.t = t
        self.name = r'$\sin({\sqrt{2}}t)$'
    def calc(self):
        return np.sin(np.sqrt(2)*self.t)
class Hamming:
    def __init__(self, N):
        self.N = N
    def calc(self):
        n = np.arange(self.N)
        return 0.54 + 0.46*np.cos((2*np.pi*n)/(self.N -1))
class sinwt:
    def __init__(self, t):
        self.t = t
        self.name = r'$\sin({1.25}t)$'
    def calc(self):
        return np.sin(1.25*self.t)
class cos3wt:
    def __init__(self, t):
        self.t = t
        self.w = 0.86
        self.name = rf'$\cos^3({self.w}t)$'
    def calc(self):
        return np.cos(self.w * self.t)**3
class cosine:
    def __init__(self, t):
        self.t = t
        self.w = 0.7
        self.d = 0.8
        self.name = r'$\cos({{\omega}_0}+{\delta})$'
    def calc(self):
        return np.cos(self.w * self.t + self.d)
class chirp:
    def __init__(self, t):
        self.t = t
        self.name = r'$\cos({16 \left({1.5} + {\frac{t}{2\pi}}\right)t})$'
    def calc(self):
        var = 16*(1.5 + (self.t/(2*np.pi)))*self.t
        return np.cos(var)

# Defining helper function
def dft(func, lim=10, start=-np.pi, end=np.pi, freq=64, plotGreen=False, Ham=False, Nois
    global x, fmax
    x = np.linspace(start, end, freq+1)[:-1]
    fmax = 1/(x[1]-x[0])
    y_ = func(x)
    if func == cosine:
        global w_act, d_act
        w_act = y_.w
        d_act = y_.d
    y_name = y_.name
    y = y_.calc()
    if Noise:
        y += 0.1*randn(freq)
    if Ham:
        h_ = Hamming(freq)
        h = fftshift(h_.calc())
        y = y*h
        y_name = y_.name + r'$\times w(t)$'
    y = fftshift(y)
    global Y, w
    Y = fftshift(fft(y))/freq
    w = np.linspace(-np.pi*fmax, np.pi*fmax, freq+1)[:-1]
    if Show:
        figure()
        subplot(2,1,1)
        plot(w,abs(Y))
        xlim([-1*lim,lim])
        ylabel(r"$|Y|$",size=16)
        title(rf"Spectrum of {y_name}")
        grid(True)
        subplot(2,1,2)
        plot(w,angle(Y),'ro')
        ii=where(abs(Y)>1e-3)
        plot(w[ii],angle(Y[ii]),'go',lw=2, visible=plotGreen)
        xlim([-1*lim,lim])
        ylabel(r"Phase of $Y$",size=16)
        xlabel(r"$k$",size=16)
        grid(True)
        show()
def weightedavg(samples, weights):
    num = np.dot(samples, weights)
    den = np.sum(weights)
    return num/den
```

0.2.2 Q1

In this Question, he has asked us to repeat the examples discussed. we have discussed about $\sin(\sqrt{2}t)$ and we will be plotting them.

```
# Q1: sin(sqrt(2)*t)
def Q1():
    '''Solves Q1'''
    # Normal DFT of sinsq2t
    dft(func=sinsq2t)
    # DFT using Hamming's window
    dft(func=sinsq2t, Ham=True, lim=8, plotRed=False)
    # DFT using Hamming's window and more samples
    dft(func=sinsq2t, Ham=True, freq=256, start=-4*np.pi, end=4*np.pi, lim=4, plotRed=Fa
```

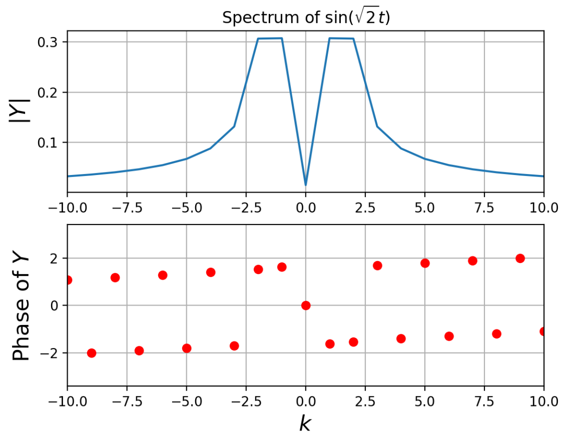


Figure 1: DFT of $\sin(\sqrt{2}t)$

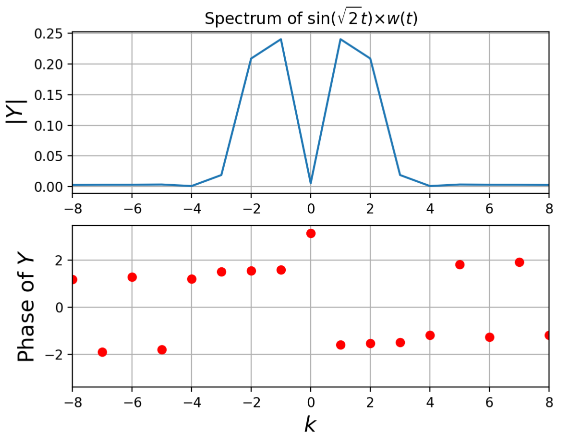


Figure 2: DFT of $\sin(\sqrt{2}t)$ with more sampling

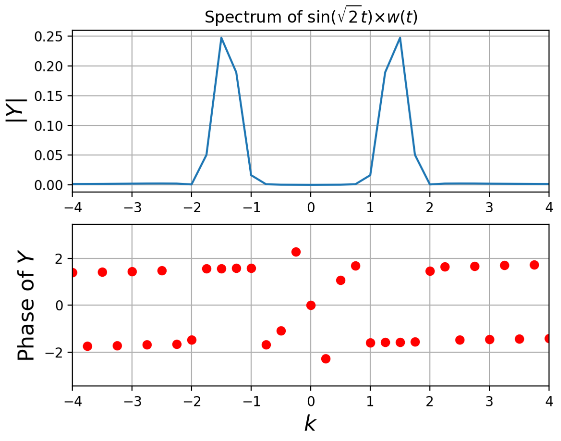


Figure 3: DFT of windowed $\sin(\sqrt{2}t)$ with high sampling

0.2.3 Q2

We are asked to find DFT of $\cos^3(w_0t)$ for w_0=0.86 with and without windowing, it can be done by the following code

```
# Q2: cos^3(w*t) where w = 0.86
def Q2():
    '''Solves Q2'''
    # Without Hamming window
    dft(func=cos3wt, start=-4*np.pi, end=4*np.pi, freq=256, Ham=False)
    # With Hamming window
    dft(func=cos3wt, start=-4*np.pi, end=4*np.pi, freq=256, Ham=True)
```

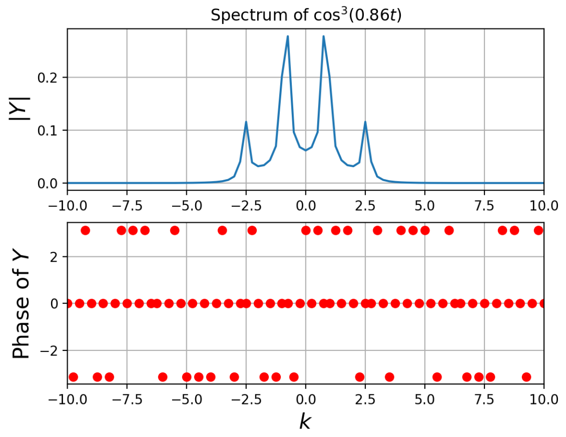


Figure 4: DFT of $\cos^3(w_0 t)$ for w_0=0.86

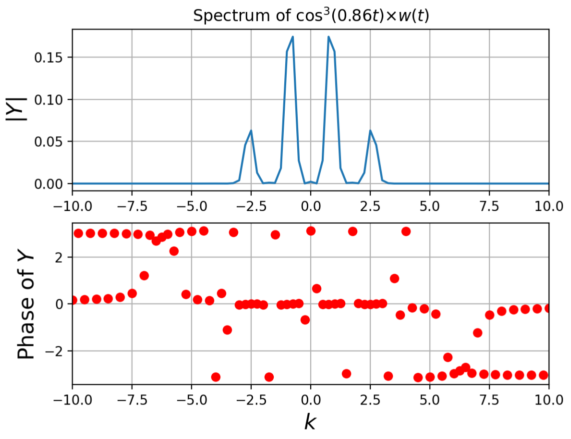


Figure 5: DFT of $\cos^3(w_0 t)$ for w_0=0.86 with windowing

0.2.4 Q3

In this section, we are asked to estimate the frequency and phase of the signal $\cos(\omega_0 t + \delta)$ given it's DFT. We can do the following using the following pseudo code.

- PLOT the DFT of $\cos(\omega_0 t + \delta)$
- CALCULATE the weighted Average of ω with weights as $|Y(j\omega)|^2$. Which is our estimated value of ω_0
- CALCULATE the nearest value to our ω_0 in our ω array
- CALCULATE the phase of corresponding $Y(j\omega)$. This acts as our estimated δ

So, the code used for the above is,

```
# Q3: Estimate w, d for cos(wt+d)
def Q3():
    '''Solves Q3'''
    dft(func=cosine, start=-np.pi, end=np.pi, freq=128, lim=4, Ham=True)
    # w_0 is to be estimated as the weighted average of w's with weights as |Y(jw)|^2
    ii = np.where(w>=0)
    w_calc = weightedavg(samples=w[ii], weights=np.abs(Y[ii])**2)
    # Finding the nearest value to w_calc in w array and finding the phase of Y(jw) at t
    jj = np.argmin(np.abs(w-w_calc))
    d_calc = np.angle(Y[jj])
    print(f'Actual w used\t: {w_act}')
    print(f'Estd. w\t: {w_calc.round(3)}')
    print(f'Actual d used\t: {d_act}')
    print(f'Estd. d\t: {d_calc.round(3)}')
```

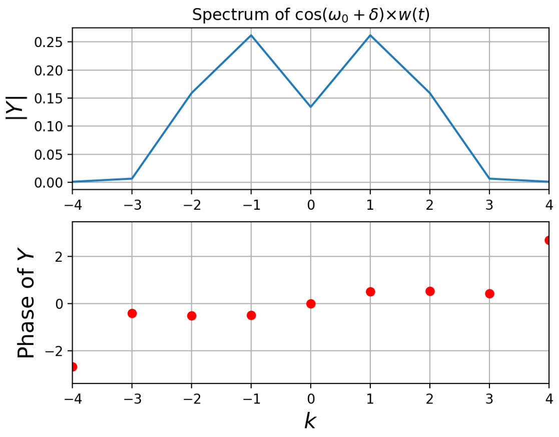
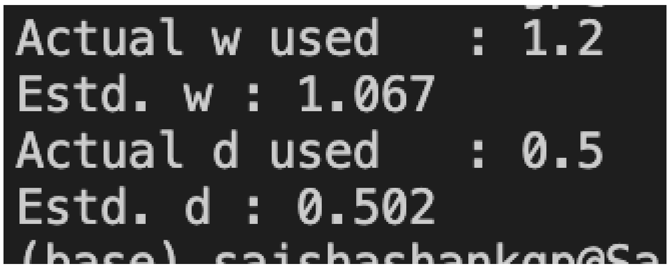


Figure 6: DFT of for $\cos(\omega_0 t + \delta)$ for ω_0=1.2 and δ = 0.5



```
Actual w used    : 1.2
Estd. w : 1.067
Actual d used    : 0.5
Estd. d : 0.502
```

Figure 7: Corresponding ω_0 and δ

0.2.5 Q4

In this section, we are asked to do the same thing as we have did in Q3 but with added white gaussian noise of max = 0.1 and mean = 0. We will be using the same procedure. The code is as follows,

```
# Q4: Estimate w, d for cos(wt+d) + white noise
def Q4():
    '''Solves Q4'''
    dft(func=cosine, start=-np.pi, end=np.pi, freq=128, lim=4, Noise=True, Ham=True)
    # w_0 is to be estimated as the weighted average of w's with weights as |Y(jw)|^2
    ii = np.where(w>=0)
    w_calc = weightedavg(samples=w[ii], weights=np.abs(Y[ii])**2)
    # Finding the nearest value to w_calc in w array and finding the phase of Y(jw) at t
    jj = np.argmin(np.abs(w-w_calc))
    d_calc = np.angle(Y[jj])
    print(f'Actual w used\t: {w_act}')
    print(f'Estd. w\t: {w_calc.round(3)}')
    print(f'Actual d used\t: {d_act}')
    print(f'Estd. d\t: {d_calc.round(3)}')
```

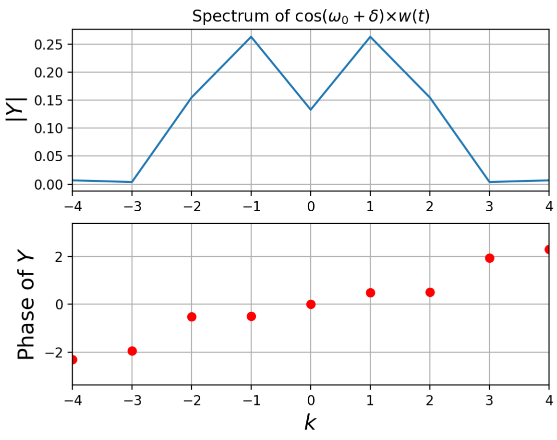
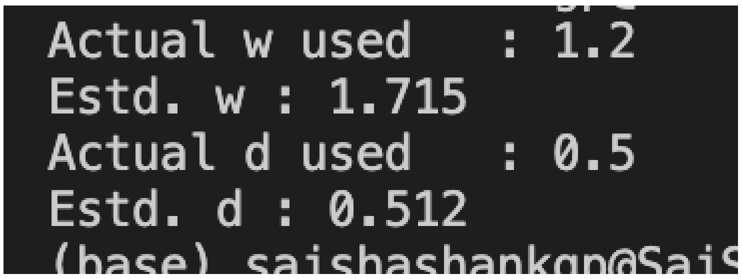


Figure 8: DFT of $\cos(\omega_0 t + \delta) + 0.1n(t)$ for ω_0=1.2 and $\delta = 0.5$



```
Actual w used    : 1.2
Estd. w : 1.715
Actual d used    : 0.5
Estd. d : 0.512
(base) saishashankgp@SaiS
```

Figure 9: Corresponding ω_0 and δ

0.2.6 Q5

We are asked to compute the DFT of a chirped signal $\cos(16\left(1.5 + \frac{t}{2\pi}\right)t)$. It is done as follows,

```
# Q5: Analysis of chirped signal
def Q5():
    '''Solves Q5'''
    # Chirped signal without windowing
    dft(func=chirp, start=-np.pi, end=np.pi, freq=1024, lim=100, Ham=False)
    # Chirped signal with windowing
    dft(func=chirp, start=-np.pi, end=np.pi, freq=1024, lim=100, Ham=True)
```

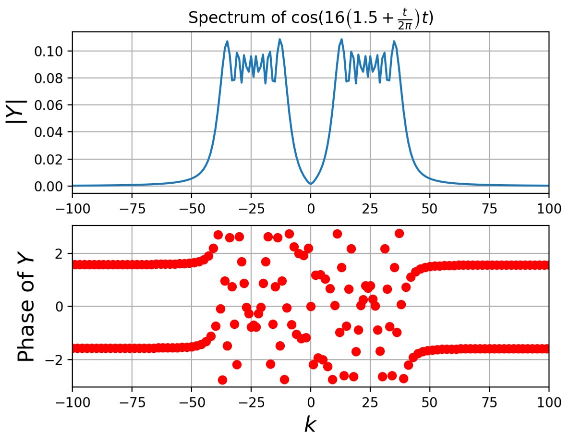


Figure 10: DFT of Chirped signal

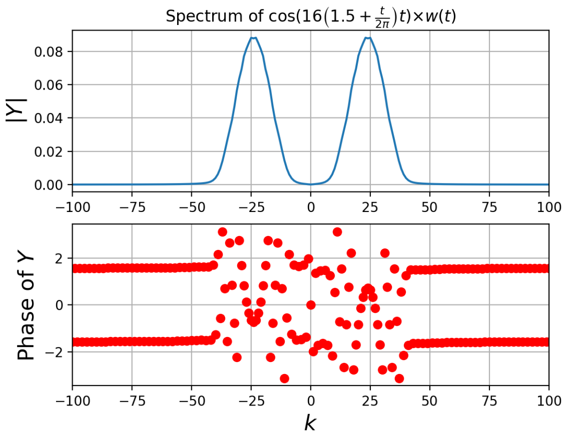


Figure 11: DFT of Chirped signal with windowing

0.2.7 Q6

Now, we are asked to plot the time-frequency dependence as a surface plot for the chirped signal. It is asked to done in regular intervals of 64 samples in time domain. So, the code is,

```
def Q6():
    '''Solves Q6'''
    dft(func=chirp, start=-np.pi, end=np.pi, freq=1024, lim=100, Ham=True, Show=False)
    t = x[::64]
    t_ = np.reshape(x, (16, 64))
    Ymag = np.zeros((16, 64))
    Yphase = np.zeros((16, 64))
    for i in range(16):
        dft(func=chirp, start=t_[i][0], end=t_[i][-1], Ham=True, freq=64, Show=False)
        Ymag[i] = np.abs(Y)
        Yphase[i] = np.angle(Y)
    w = linspace(-fmax*np.pi,fmax*np.pi,65)
    w = w[:-1]
    t,w = meshgrid(t,w)
```

```
fig = figure()
ax = fig.add_subplot(111, projection='3d')
surf=ax.plot_surface(w,t,Ymag.T,cmap='viridis',linewidth=0, antialiased=False)
fig.colorbar(surf, shrink=0.5, aspect=5)
ax.set_title('surface plot')
ylabel(r"$\omega\rightarrow$")
xlabel(r"$t\rightarrow$")
show()
```

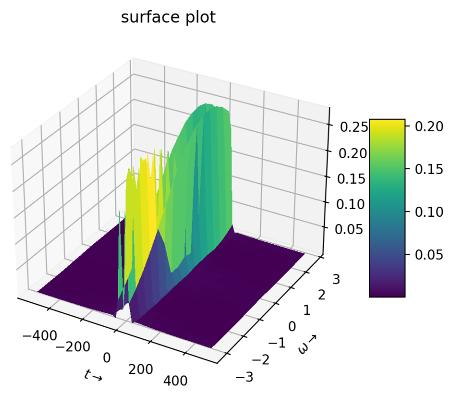


Figure 12: Surface plot of time-frequency dependence of DFT